gpusat2 – An Improved GPU Model Counter

Johannes K. Fichte[1] Markus Hecher[2,3] Markus Zisser[2]

[1]TU Dresden, Germany
[2]TU Wien, Austria
[3]University of Potsdam, Germany

Pragmatics of SAT (POS) Workshop 2019,
Lisbon, Portugal

July 8, 2019

Motivation

Model Counting (#SAT)

- Generalizes Boolean satisfiability problem (SAT)
- #SAT: output the number of satisfying assignments
- WMC: output the weighted model count
- Various applications in AI and reasoning, e.g.,
  - Bayesian reasoning [Sang et al.'05]
  - Learning preference distributions [Choi et al.'15]
  - Infrastructure reliability [Meel et al.17]
- Computational complexity: #P-hard [Roth'96]

Motivation: A somewhat different approach.

#SAT/WMC Solving

- There are already plenty solvers based on various techniques: approximate (Meel) / CDCL (Baccus/Thurley) / knowledge compilation based (Darwiche et al.)

Parameterized Algorithms

- Lots of theoretical work over last 20 years and various algorithms for #SAT

Research Question

Are (theoretical) algorithms from parameterized complexity even useful for implementations in #SAT/WMC solving?

Motivation: A somewhat different approach.

#SAT/WMC Solving

- There are already plenty solvers based on various techniques:
  approximate (Meel) / CDCL (Baccus/Thurley) /
  knowledge compilation based (Darwiche et al.)

Parameterized Algorithms

- Lots of theoretical work over last 20 years and various algorithms for #SAT

Research Question

Are (theoretical) algorithms from parameterized complexity even useful for implementations in #SAT/WMC solving?

Parameterized Algorithmics

Topic of the Talk

Solve #SAT/WMC by means of an implementation of a parameterized algorithm that explicitly exploits small treewidth.

Presentation:

1. Ideas towards a GPU model counter [FHWoltranZ'18]
2. Improved Architecture for #SAT (POS paper [FHZ'19])

Purpose:

There are other architectures out there and it might fit for certain algorithms.
NOT: outperforming everything else.

Parameterized Algorithmics

Topic of the Talk

Solve #SAT/WMC by means of an implementation of a parameterized algorithm that explicitly exploits small treewidth.

Presentation:

1. Ideas towards a GPU model counter [FHWoltranZ'18]
2. Improved Architecture for #SAT (POS paper [FHZ'19])

Purpose:

There are other architectures out there and it might fit for certain algorithms.

NOT: outperforming everything else.

Topic of the Talk

Solve #SAT/WMC by means of an implementation of a parameterized algorithm that explicitly exploits small treewidth.

Presentation:

1. Ideas towards a GPU model counter [FHWoltranZ'18]
2. Improved Architecture for #SAT (POS paper [FHZ'19])

Purpose:

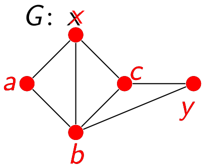
There are other architectures out there and it might fit for certain algorithms.

NOT: outperforming everything else.

Tree Decompositions

Treewidth ▸ Definition & Example

- Most prominent graph invariant
- Small treewidth indicates tree-likeness and sparsity
- Can be used to solve #SAT/WMC by defining graph representations of the input formula

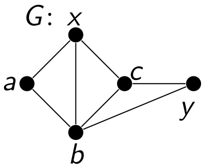


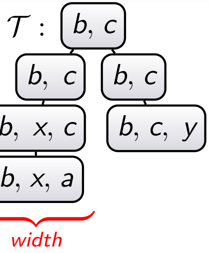
Treewidth

Definition & Example

- Treewidth defined in terms of tree decompositions (TD)
- TD: arrangement of graph into a tree + bags s.t. ...
- Treewidth: width of a TD of smallest width

Tree Decompositions



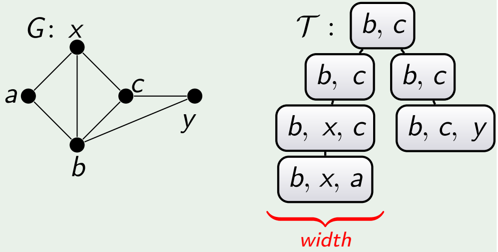


Treewidth ▸ Definition & Example

- Treewidth defined in terms of tree decompositions (TD)
- TD: arrangement of graph into a tree + bags s.t. ...
- Treewidth: width of a TD of smallest width

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G

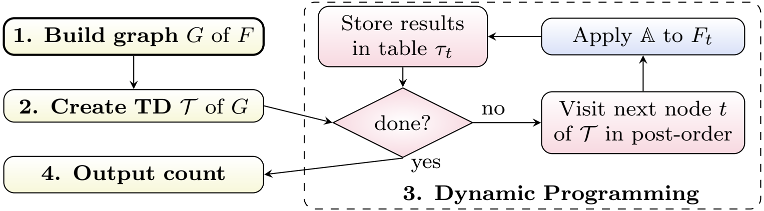


Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. Each vertex must occur in some bag
2. For each edge, there is a bag containing both endpoints
3. Connected: If vertex v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Outline (Basic Architecture)



Part:

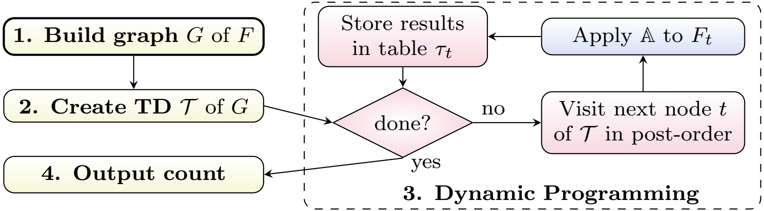
A) Background & Basic Concepts

Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA'10]

B) Finding TDs (2)

C) Dynamic Programming (3) on the GPU

Outline (Basic Architecture)



Part:

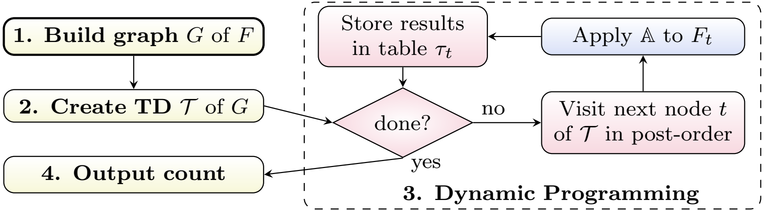
A) Background & Basic Concepts

Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA'10]

B) Finding TDs (2)

C) Dynamic Programming (3) on the GPU

Outline (Basic Architecture)



Part:

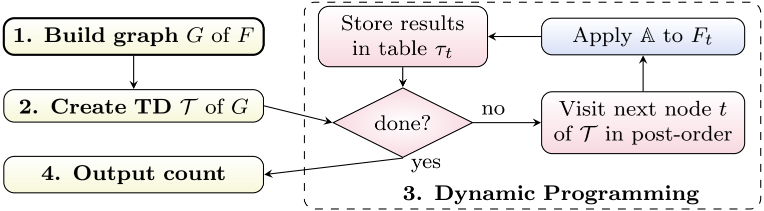
A) Background & Basic Concepts

Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA'10]

B) Finding TDs (2)

C) Dynamic Programming (3) on the GPU

Outline (Basic Architecture)



Part:

A) Background & Basic Concepts

Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA'10]

B) Finding TDs (2)

C) Dynamic Programming (3) on the GPU

How to “use” tree decompositions for #SAT/WMC?

Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

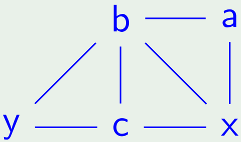
$$Mod(\varphi) = \{ \{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}, \{b, c, x\}, \{a, b, c, x\}, \{b, y\}, \{a, b, y\}\}$$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions

Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y)$

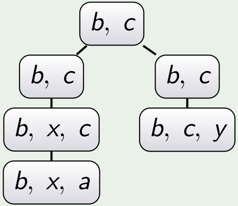
1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

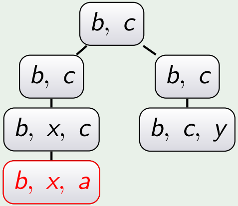
1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



b	c
1	0
1	1

b	c
1	0
1	1

b	c
0	0
1	0
1	1

b	x	c
1	0	0
1	0	1
1	1	1

b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

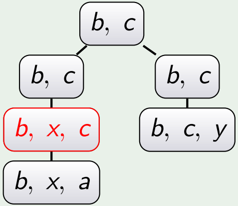
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

"Local formula" F_t clauses whose variables are contained in the bag (colored in red above)

Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



b	c
1	0
1	1

b	c
1	0
1	1

b	c
0	0
1	0
1	1

b	x	c
1	0	0
1	0	1
1	1	1

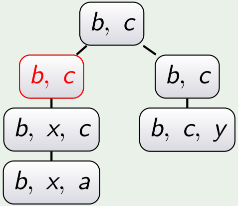
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



b	c
1	0
1	1

b	c
1	0
1	1

b	c
0	0
1	0
1	1

b	x	c
1	0	0
1	0	1
1	1	1

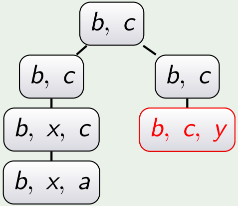
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



b	c
1	0
1	1

b	c
1	0
1	1

b	x	c
1	0	0
1	0	1
1	1	1

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

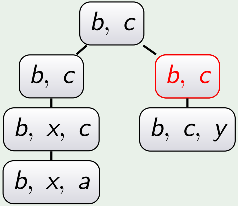
b	c
0	0
1	0
1	1

b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



b	c
1	0
1	1

b	c
1	0
1	1

b	x	c
1	0	0
1	0	1
1	1	1

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

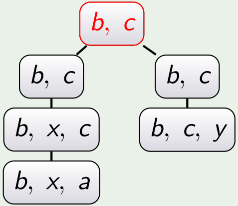
b	c
0	0
1	0
1	1

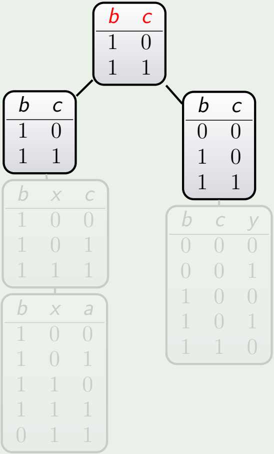
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions

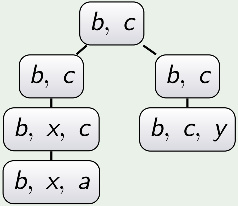


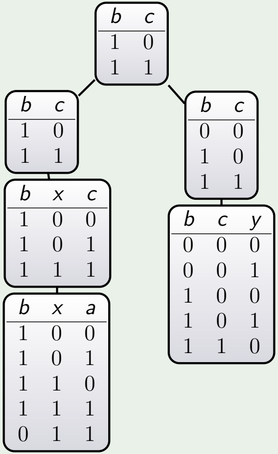


Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions

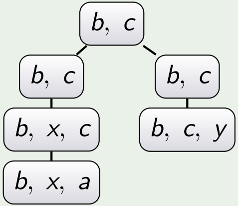


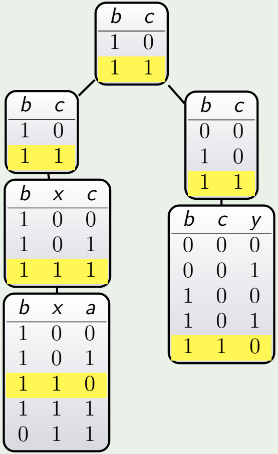


Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions

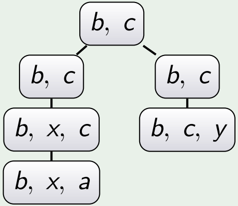




Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



b	c	#
1	0	4
1	1	4

b	c	#
1	0	2
1	1	4

b	c	#
0	0	2
1	0	2
1	1	1

b	x	c	#
1	0	0	2
1	0	1	2
1	1	1	2

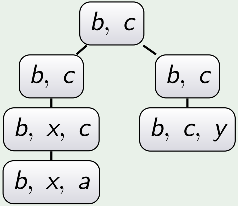
b	c	y	#
0	0	0	1
0	0	1	1
1	0	0	1
1	0	1	1
1	1	0	1

b	x	a	#
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1
0	1	1	1

Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



b	c	#
1	0	4
1	1	4

b	c	#
1	0	2
1	1	4

b	x	c	#
1	0	0	2
1	0	1	2
1	1	1	2

b	x	a	#
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1
0	1	1	1

b	c	#
0	0	2
1	0	2
1	1	1

b	c	y	#
0	0	0	1
0	0	1	1
1	0	0	1
1	0	1	1
1	1	0	1

Runtime: $2^{O(tw)} \cdot \text{poly}(|\varphi|)$

“Find” tree decompositions of small width?

Works well even for relatively large instances.

Thanks to the Parameterized Algorithms and Computational Experiments Challenge (PACE) ’16/’17!!!

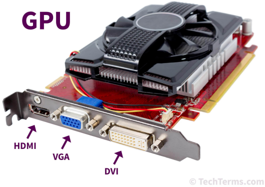
“Find” tree decompositions of small width?

Works well even for relatively large instances.

Thanks to the Parameterized Algorithms and Computational Experiments Challenge (PACE) ’16/’17!!!

A GPU-based #SAT/WMC-solver

OR how to go parallel?



How to parallelize DP?

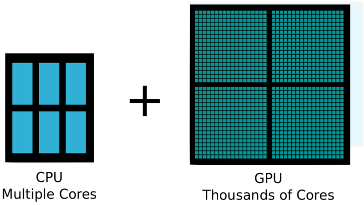
1. Compute tables for multiple nodes in parallel

$\Rightarrow$ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

$\Rightarrow$ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Dynamic Programming on the GPU

How to parallelize DP?

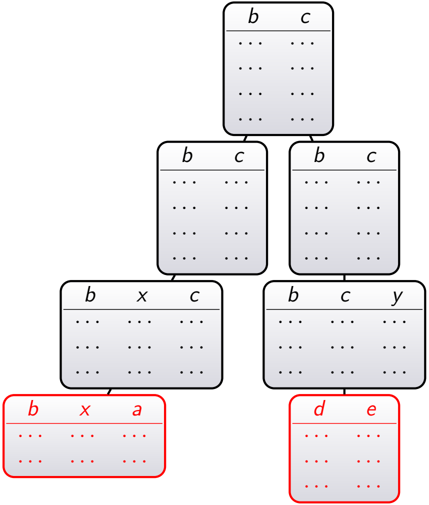
1. Compute tables for multiple nodes in parallel

$\Rightarrow$ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

$\Rightarrow$ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Dynamic Programming on the GPU

How to parallelize DP?

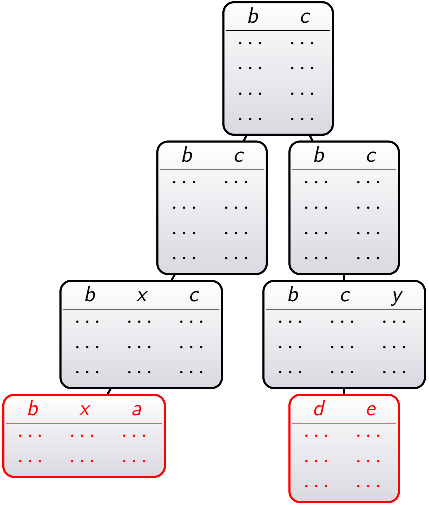
1. Compute tables for multiple nodes in parallel

$\Rightarrow$ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

$\Rightarrow$ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Dynamic Programming on the GPU

How to parallelize DP?

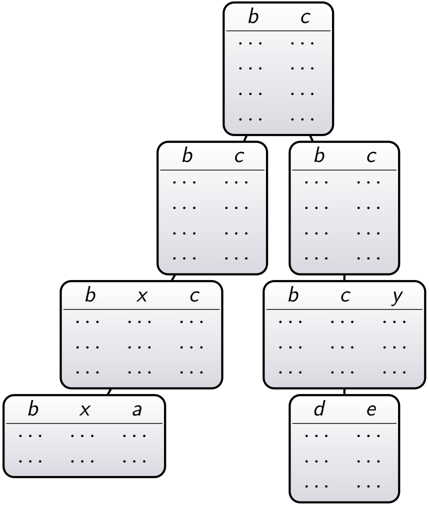
1. Compute tables for multiple nodes in parallel

$\Rightarrow$ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

$\Rightarrow$ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Dynamic Programming on the GPU

How to parallelize DP?

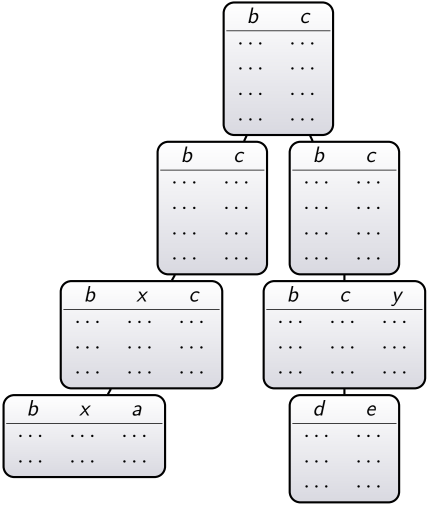
1. Compute tables for multiple nodes in parallel

$\Rightarrow$ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

$\Rightarrow$ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Implementation

Disclaimer for theorists: you need to get your hands dirty

$+$

Right hindsight

Implementation Ideas

Right hindsight?

1. Data structures: a "pixel" represents #solutions store data as
   a. Array (gpuSAT1); improved in gpuSAT2
   b. Compressed partial assignments in BST (gpuSAT2)
2. Avoid Copying:
   Merge small bags (gpuSAT1 < 14, gpuSAT2 hardware dep.)
3. Handle potential VRAM overflow (gpuSAT2):
   Split bags and previously computed solutions
   (if 2^w assignments do not fit into the VRAM)
4. Get counters right

Implementation Ideas (cont.)

(1) Data Structures

a. Array: memory address (plus offset) identifies assignment

$\Rightarrow$ Issue: produces lots of memory cells that contain value 0

b. BST (gpuSAT2):

- Compress Assignments (or address assignments not just by a memory cell)
- Store only where $\# \neq 0$
- Idea: use BST; simulate this in an array
  (implement manually on GPU; no libs)

Implementation Ideas (cont).

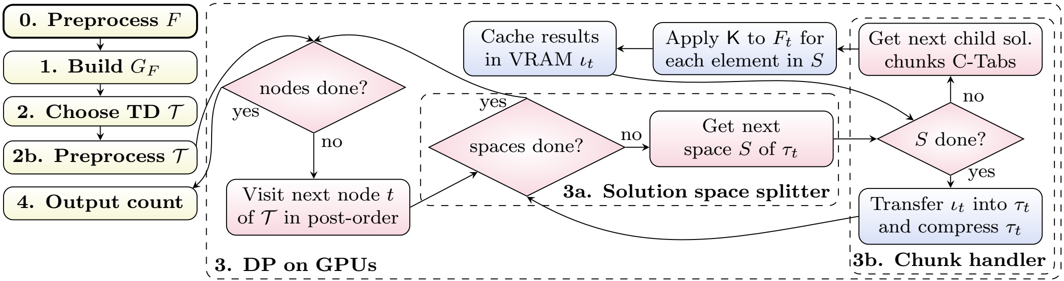
(4) Counters:

- WMC: double or double4 (gpuSAT1)
- #SAT
  a. run WMC and use uniform factor (gpuSAT1)
  b. use logarithmic counters (gpuSAT2)
     - Store floating log-counters
     - Numbers stored in relation to exponent 2^e (largest exponent)
     - Dynamically change exponent (keep highest possible precision)

In Practice

- Available on github (GPL3)
- OpenCL: vendor and hardware independent computation framework; C++11
- Works for two graph types: primal, incidence, dual graph

New Architecture (gpuSAT2) [FHZ19]

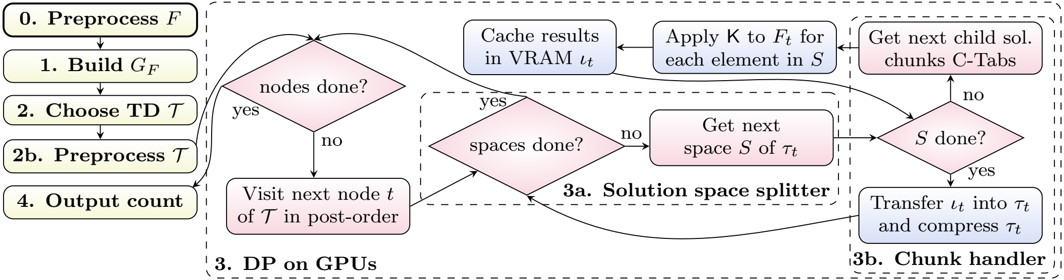


0. Instance Preprocessing
2. Customized Tree Decompositions

3a. Solution Space Splitting

3b. Execute a small GPU-program in a GPU thread (kernel) for each element in S
Compress the data and store it in the VRAM (separate GPU-programs)
After all chunks are processed memory regions are merged

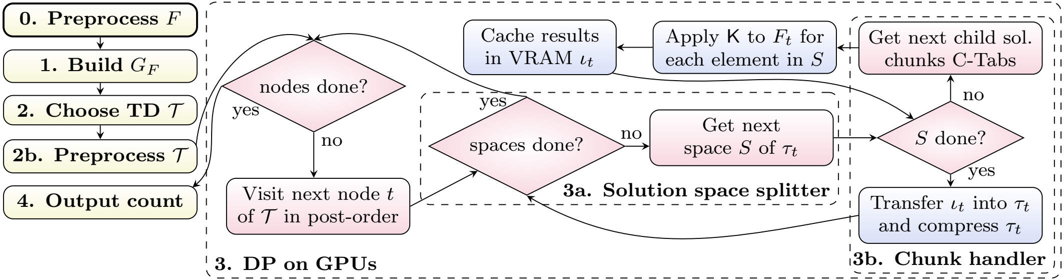
New Architecture (gpuSAT2) [FHZ19]



0. Instance Preprocessing
2. Customized Tree Decompositions
   (#30; minimize max. card. of intersection of bags at node and its children)

3a. Solution Space Splitting
3b. Execute a small GPU-program in a GPU thread (kernel) for each element in S
   Compress the data and store it in the VRAM (separate GPU-programs)
   After all chunks are processed memory regions are merged

New Architecture (gpuSAT2) [FHZ19]

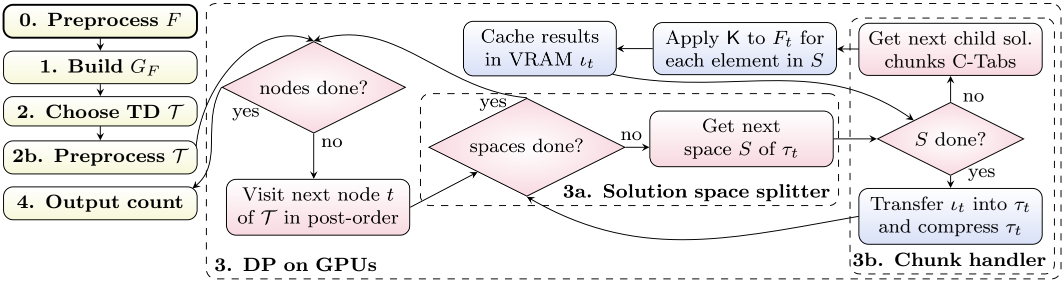


0. Instance Preprocessing
2. Customized Tree Decompositions

3a. Solution Space Splitting
(Split larger solutions into smaller portions ⇒ avoid OOM)

3b. Execute a small GPU-program in a GPU thread (kernel) for each element in S
Compress the data and store it in the VRAM (separate GPU-programs)
After all chunks are processed memory regions are merged

New Architecture (gpuSAT2) [FHZ19]



0. Instance Preprocessing
2. Customized Tree Decompositions
3a. Solution Space Splitting
3b. Execute a small GPU-program in a GPU thread (kernel) for each element in S
    Compress the data and store it in the VRAM (separate GPU-programs)
    After all chunks are processed memory regions are merged

Experimental Work

Instances

- 2585 instances from public benchmarks
- #SAT and WMC

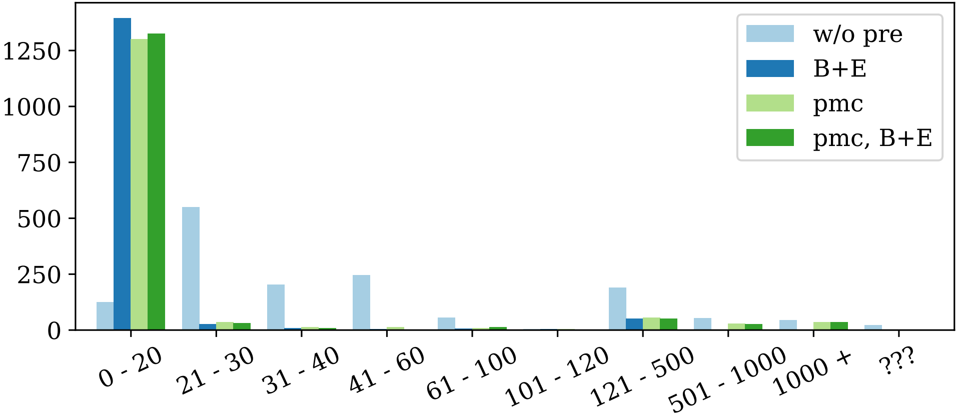
Limits

- Cannot expect to solve instances of high treewidth.

Experiments

1. Distribution of width
2. Benchmarked all solvers that are publicly available

#SAT: Width Comparison (Preprocessing comp.)

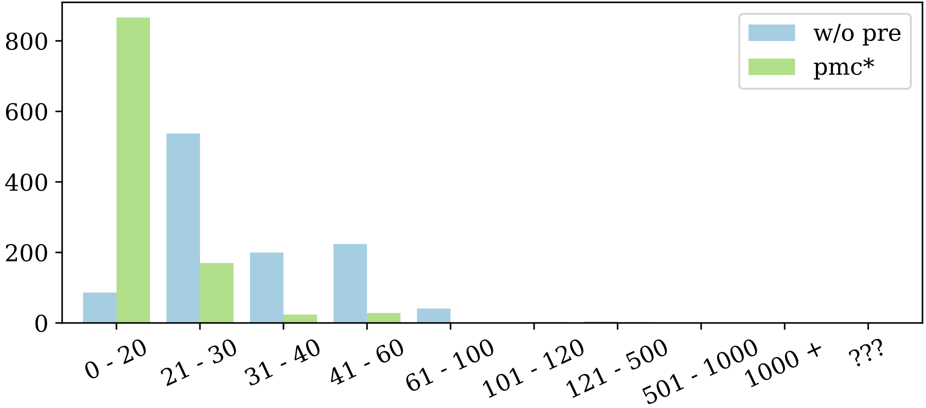


- Runtime well below a second (max. 2.5) 0–40; timeout (900s) on 41

⇒ 54% primal treewidth below 30; 70% below 40

⇒ Preprocessing produces TDs of significantly smaller width

WMC: Width Comparison (w/o Preprocessing)



⇒ Produce decompositions of significantly smaller width

Experimental Work (Runtime)

Setting (Runtime Comparision)

Take gpuSAT1, gpuSAT2, and versions as well as sequential and parallel solvers. Consider Wallclock

Hardware

- non-GPU solving: cluster of 9 nodes; each E5-2650 CPUs(12cores) 2.2 GHz, 256 GB RAM; disabled HT, kernel 4.4
- GPU-solving: i3-3245 3.4 GHz; 16 GB RAM; GPU: Sapphire Pulse ITX Radeon RX 570 GPU; 1.24 GHz with 32 compute units, 2048 shader units, 4GB VRAM

Experimental Work (Runtime Disclaimer)

Questionable Setting?

Aren't you comparing apples and oranges? YES.

Problems of the Setting

- We compare on different hardware

$\Rightarrow$ Soon, new cluster node with the same specs and two GPUs

- Wallclock is unfair.

  Usually user is interested in getting things done quickly (+ fairly cheap)

$\Rightarrow$ Power consumption (Joule) and price of investment better measure (BUT not accessible with the current framework)

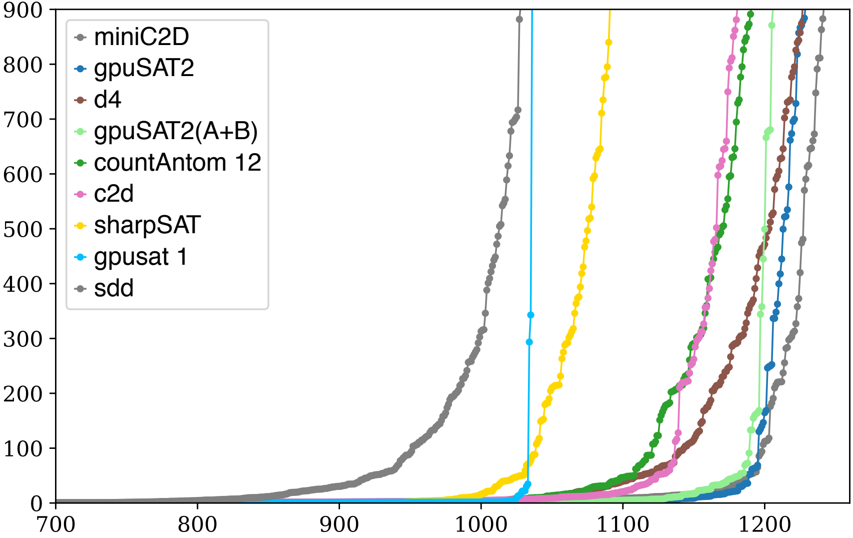
$\Rightarrow$ We use cheap consumer hardware (200 EUR) for the GPU not a Tesla K80 (8k EUR) or DGX2 (400k EUR)

- Parallel vs. sequential: No excuse, sorry

#SAT

	solver	0-20	21-30	31-40	41-50	51-60	>60	best	unique	$\sum$	time[h]
pmc preprocessing	miniC2D	1193	29	**10**	2	1	7	13	0	**1242**	**68.77**
	gpuSAT2	**1196**	**32**	1	0	0	0	250	**8**	1229	71.27
	d4	1163	20	**10**	2	**4**	28	52	1	1227	76.86
	gpuSAT2(A+B)	1187	18	1	0	0	0	120	7	1206	74.56
	countAntom 12	1141	18	**10**	**5**	**4**	13	101	0	1191	84.39
	c2d	1124	31	**10**	3	3	10	20	0	1181	84.41
	sharpSAT	1029	16	**10**	2	**4**	**30**	**253**	1	1091	106.88
	gpuSAT1	1020	16	0	0	0	0	106	7	1036	114.86
	sdd	1014	4	7	1	0	2	0	0	1028	124.23
	solver	0-20	21-30	31-40	41-50	51-60	>60	best	unique	$\sum$	time[h]
without preprocessing	countAntom 12	118	511	139	**175**	**21**	**181**	318	15	**1145**	**96.64**
	d4	124	514	148	162	**21**	168	69	15	1137	104.94
	c2d	119	525	**165**	161	18	120	48	15	1108	110.53
	miniC2D	122	514	128	149	9	62	0	0	984	141.22
	sharpSAT	100	467	124	156	12	123	**390**	4	982	135.41
	gpuSAT2(A+B)	**125**	**539**	96	138	0	0	94	**19**	898	151.16
	gpuSAT2	**125**	523	96	138	0	0	78	17	882	155.43
	gpuSAT1	**125**	524	67	140	0	0	82	9	856	162.03
	cachet	99	430	71	152	8	57	3	0	817	176.26
	solver	0-20	21-30	31-40	41-50	51-60	>60	best	unique	$\sum$	time[h]

#SAT: Runtime Results (w. Preprocessing)



⇒ Techniques pay off after preprocessing

WMC

	solver	0-20	21-30	31-40	41-50	51-60	>60	best	unique	$\sum$	time[h]
with pmc*	miniC2D	858	**164**	**6**	**0**	**0**	3	13	**8**	**1031**	21.29
	gpuSAT1	**866**	158	0	**0**	**0**	0	**348**	4	1024	18.03
	gpuSAT2(A+B)	**866**	156	0	**0**	**0**	0	343	4	1022	**17.86**
	gpuSAT2	**866**	138	0	**0**	**0**	0	299	4	1004	22.43
	d4	810	106	0	**0**	**0**	0	46	0	916	55.36
	cachet	617	128	1	**0**	**0**	3	106	1	749	93.65
without pre	d4	82	501	**142**	**156**	**10**	**19**	111	**24**	**910**	**53.97**
	miniC2D	84	517	134	152	3	4	19	7	894	59.69
	gpuSAT2(A+B)	**86**	**527**	98	138	0	0	167	19	849	64.40
	gpuSAT2	**86**	511	98	138	0	0	131	7	833	68.61
	gpuSAT1	**86**	513	68	140	0	0	**182**	10	807	73.78
	cachet	60	447	100	145	2	9	118	1	763	89.80

Summary

Contributions

- Established Architecture for DP on the GPU
- Competitive Implementation for #SAT/WMC solving

Benchmark: Comparing apples and oranges

BUT: you compare parallel and sequential solvers.

1. We run on cheap consumer hardware (200 EUR).
2. Cannot measure speedup due to OpenCL limitations
   $\Rightarrow$ migrate to cuda

Summary contd.

Take Home Messages

1. Parameterized Algorithms can actually work
   (Preprocessing is key; some techniques pay only off with right preprocessing)
2. Does it work for SAT? $\Rightarrow$ we don't expect so.

Future Work

- Improve current setup by:
  Portfolio solving; Parallel Usage of GPUs; Alternative Frameworks
- Consider whether stable among different GPU hardware
- Parameters (pswidth)

Sponsors: FWF Y698 & P26696; DFG HO 1294/11-1

Summary contd.

Take Home Messages

1. Parameterized Algorithms can actually work
   (Preprocessing is key; some techniques pay only off with right preprocessing)
2. Does it work for SAT? $\Rightarrow$ we don't expect so.

Future Work

- Improve current setup by:
  Portfolio solving; Parallel Usage of GPUs; Alternative Frameworks
- Consider whether stable among different GPU hardware
- Parameters (pswidth)

Thanks for listening!

Sponsors: FWF Y698 & P26696; DFG HO 1294/11-1

References

[AMW17]: Abseher, Musliu, Woltran. htd – A Free, Open-Source Framework for (Customized) Tree Decompositions and Beyond. CPAIOR'17. 2017. doi: 10.1007/978-3-319-59776-8_30

[FHWZ18]: Fichte, Hecher, Woltran, Zisser. Weighted Model Counting on the GPU by Exploiting Small Treewidth. ESA'18. 2018. doi: 10.4230/LIPIcs.ESA.2018.28

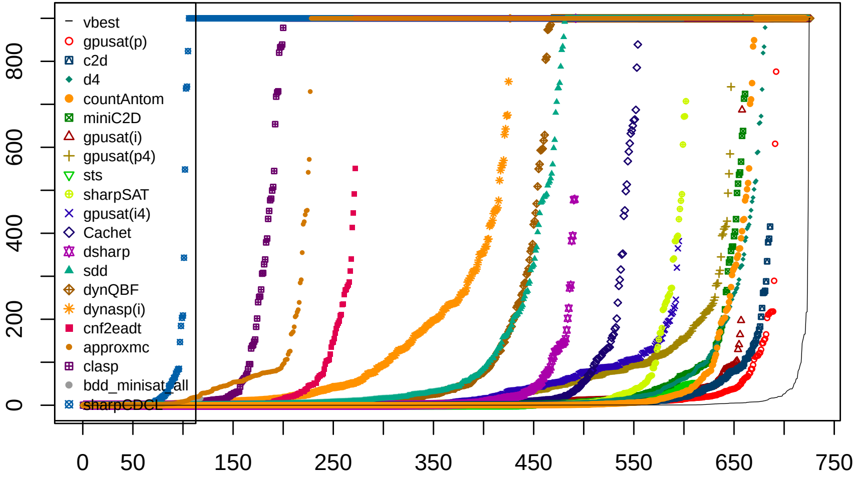
[FHZ19]: Fichte, Hecher, Zisser. gpusat2 – An Improved GPU Model Counter. POS 2019.

[SamerSzeider10]: Samer, Szeider. Algorithms for propositional model counting. JDA. 2010. doi: 10.1016/j.jda.2009.06.002

gpusat is available at: https://github.com/daajoe/gpusat

Backup Slides

Solving (Width: 0–30): #SAT



kc/cdcl: c2d, d4, dsharp

dp: gpusat, dynQBF, dynasp

parallel: countAntom, gpusat

cdcl: Cachet, sharpSAT, clasp

bdd: sdd

approx: approxmc, sts

Solving: #SAT

solver	0-20	21-30	31-40	41-50	51-60	>60	best	$\sum$	rank
c2d	164	519	**175**	116	20	118	120	1112	2
Cachet	133	421	91	109	8	58	13	820	7
d4	**169**	510	156	119	**23**	**162**	191	**1139**	1
gpusat(p)	**169**	**523**	79	104	0	0	88	875	6
miniC2D	167	491	137	103	8	67	2	973	4
sharpSAT	136	465	136	112	11	124	**483**	984	3
sts	162	448	101	**146**	10	45	252	912	5

Table: Number of counting instances solved by solver and interval.

Empirical Work (first approach)

Observations

- Implementation is fairly naive
- Still: competitive up to width 30
- Requirement: obtain decompositions fast
- Width was surprisingly small (different for SAT)

Implementation Ideas (cont.)

(1) Data Structures

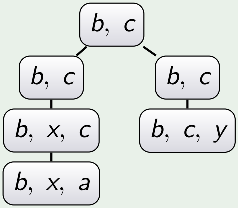
b. BST (details):

- Continuous sequence 64-bit unsigned integers (cells)
- Cell: empty, index, and value (counter)
- index cells: lower 32 bits index to the next cell
  (lower bits assignment 0, upper 1)
- Handle Sync (between parallel threads) by keeping track of the current size
  (number of allocated cells; prevent to allocate cell again)

Solving #SAT [SamerSzeider10]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $\mathcal{S}$
4. Combine solutions



"Local formula" F_t clauses whose variables are contained in the bag (colored in red above)

Nice Tree Decompositions
(note example left is not nice)

LEAF.: Put empty set and counter 1

INTR.: Guess truth value and check satisfiability

REMOVE: Remove *a* from each assignment (row) in the table and sum up the counters if we get multiple assignments with the same data

JOIN: Match rows with the same assignment and multiply the counters

Algorithm for Primal Graph

In: Node t, bag χ_t, clauses F_t, sequence C of tables.

Out: Table tab_t

if $\mathrm{type}(t) = \mathrm{leaf}$ **then**

  $\mathrm{tab}_t \leftarrow \{\emptyset\}$

else if $\mathrm{type}(t) = \mathrm{intr}$ *and* $a \in \chi_t \setminus \chi_{t'}$, **then**

$$\mathrm{tab}_t \leftarrow \left\{ \tau \cup \{a\} \quad \middle| \; \tau \in \mathrm{tab}'', \tau \cup \{a\} \models F_t \right\} \cup$$

$$\left\{ \tau \quad \middle| \; \tau \in \mathrm{tab}'', \tau \models F_t \right\}$$

else if $\mathrm{type}(t) = \mathrm{rem}$ *and* $a \in \chi_{t'} \setminus \chi_t$ **then**

$$\mathrm{tab}_t \leftarrow \left\{ \tau \setminus \{a\} \quad \middle| \; \tau \in \mathrm{tab}'' \right\}$$

else if $\mathrm{type}(t) = \mathrm{join}$ **then**

$$\mathrm{tab}_t \leftarrow \left\{ \tau \quad \middle| \; \tau \in \mathrm{tab}'', \tau \in \mathrm{tab}'' \right\}$$

return tab_t